

Presentación para usar con el libro de texto, *Algorithm Design and Applications*, por MT Goodrich y R. Tamassia, Wiley, 2015

Análisis de Algoritmos



Entrada



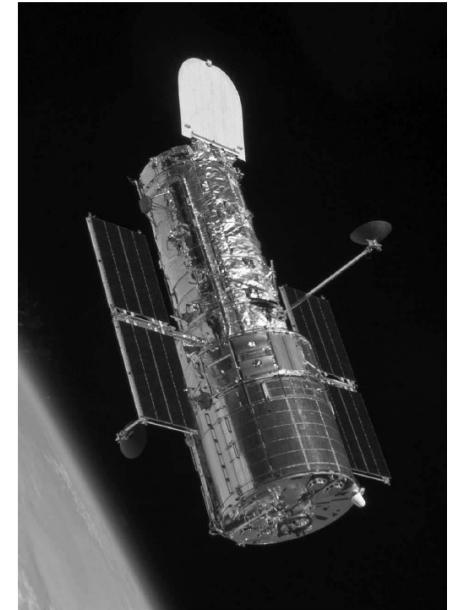
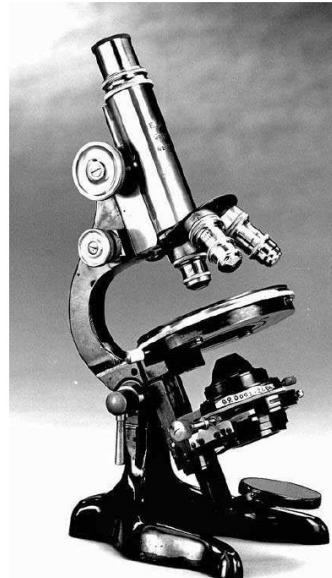
Algoritmo



Salida

Escalabilidad

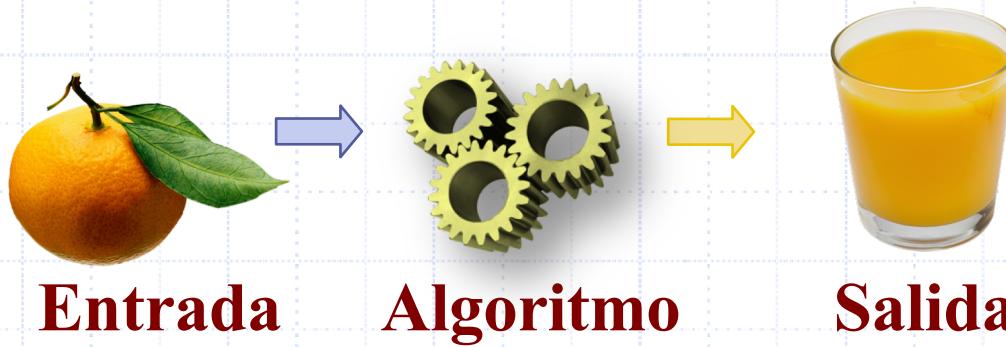
- Los científicos a menudo tienen que lidiar con diferencias de escala, desde lo microscópicamente pequeño hasta lo astronómicamente grande.
- Los informáticos también deben lidiar con la escala, pero lo hacen principalmente en términos de **volumen de datos** en lugar de tamaño de objeto físico.
- **Escalabilidad** se refiere a la capacidad de un sistema para adaptarse correctamente tamaños crecientes de entradas o cantidades de carga de trabajo (workload)



Microscope: U.S. government image, from the N.I.H. Medical Instrument Gallery, DeWitt Stetten, Jr., Museum of Medical Research. Hubble Space Telescope: U.S. government image, from NASA, STS-125 Crew, May 25, 2009.

Algoritmos y estructuras de datos

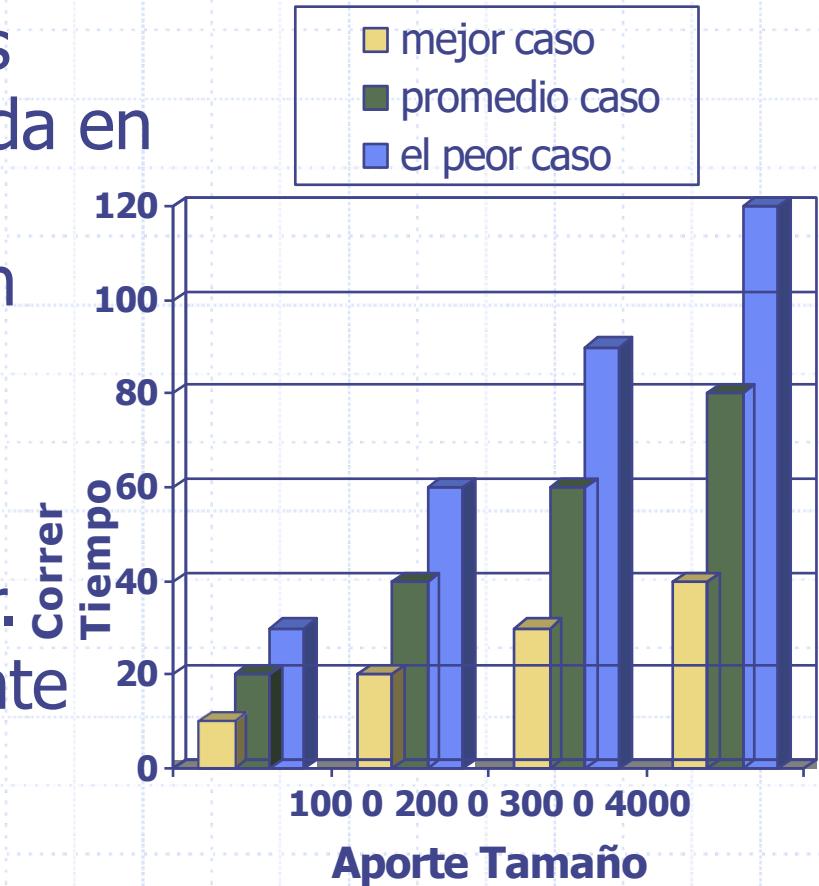
- Un **algoritmo** es un procedimiento paso-a-paso para realizar alguna tarea en un tiempo finito.
 - Por lo general, un algoritmo toma datos de entrada y produce una salida basada en él.



- Una **estructura de datos** es una forma sistemática de organizar y acceder datos.

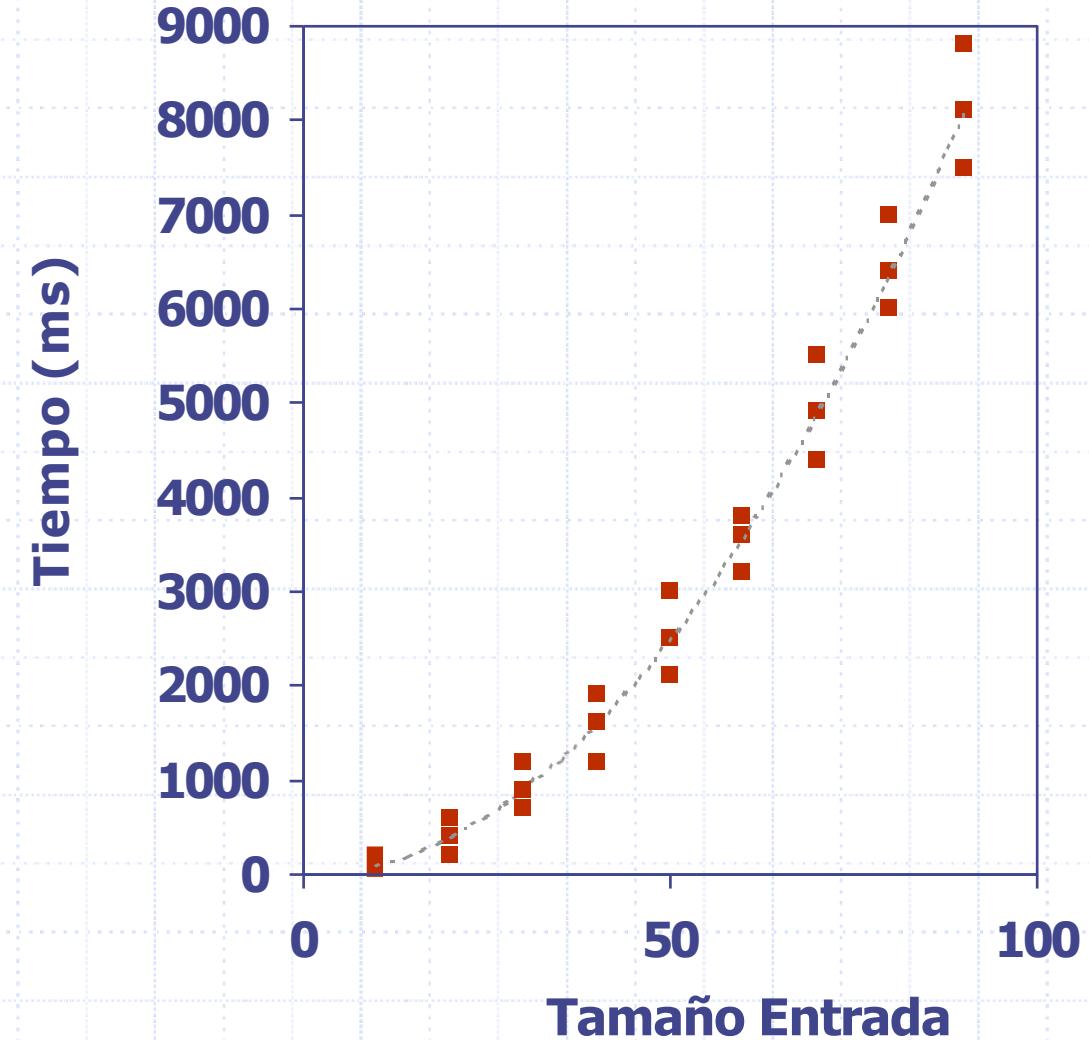
Tiempo de Ejecución

- La mayoría de los algoritmos transforma objetos de entrada en objetos de salida
- El tiempo de ejecución de un algoritmo típicamente crece con el tamaño de la entrada
- El tiempo promedio es a menudo difícil de determinar.
- Nos enfocamos principalmente en el **peor de los casos**
 - más fácil de analizar
 - Crucial para aplicaciones como juegos, finanzas y robótica



Estudio Experimental

- Escribe un programa implementando el algoritmo
- Ejecutar el programa con entradas de diferentes tamaños
- Graficar los resultados

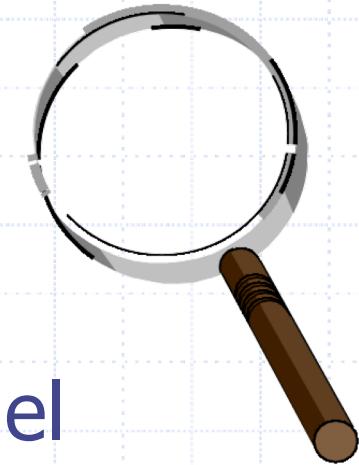


Limitaciones Experimentales

- Es necesario implementar el algoritmo, que puede ser difícil
- Los resultados pueden no ser indicativos del tiempo de ejecución en otras entradas no incluidas en el experimento.
- Para comparar dos algoritmos, se deben utilizar los mismos entornos de hardware y software.



Análisis Teórico



- Utiliza una descripción de alto nivel del algoritmo en lugar de una implementación .
- Caracteriza el tiempo de ejecución en función del tamaño de entrada, n
- Tiene en cuenta todos las posibles entradas
- Nos permite evaluar la velocidad de un algoritmo independientemente del entorno de hardware/software

Pseudocódigo

- Descripción de alto nivel de un algoritmo
- Más estructurado que un idioma humano
- Menos detallado que un programa.
- Notación preferida para describir algoritmos
- Oculta el diseño del programa asuntos

Pseudocódigo Detalles

- Control de Flujo
 - `if ... then ... [else ...]`
 - `while ... do ...`
 - `repeat ... until ...`
 - `for ... do ...`
- Declaración de método

Algorithm **method (arg [, arg...])**

Input ...

Output ...
- Llamada a un Método

method (arg [, arg...])
- Devolver valor

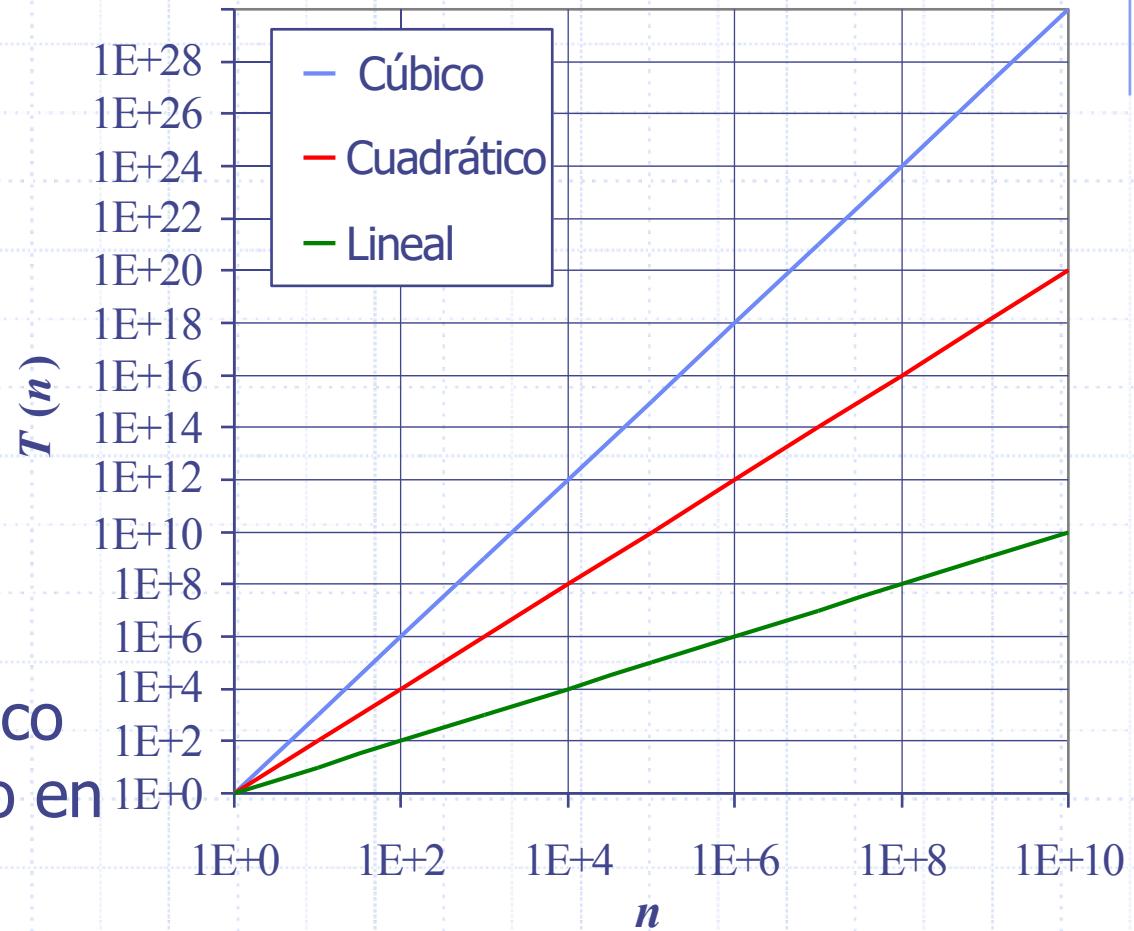
`return expression`
- Expresiones:
 - ← Asignación
 - $= =$ Pruebas de Igualdad
 - n^2 superíndices y otros formalismos matemáticos

Siete Funciones Importantes

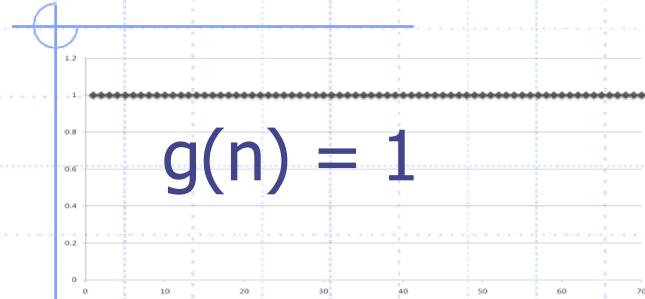
- Siete funciones que aparecen a menudo en el análisis de algoritmos:

- constante ≈ 1
- Logarítmico $\approx \log n$
- Lineal $\approx n$
- N-Log-N $\approx n \log n$
- Cuadrático $\approx n^2$
- Cúbico $\approx n^3$
- exponencial $\approx 2^n$

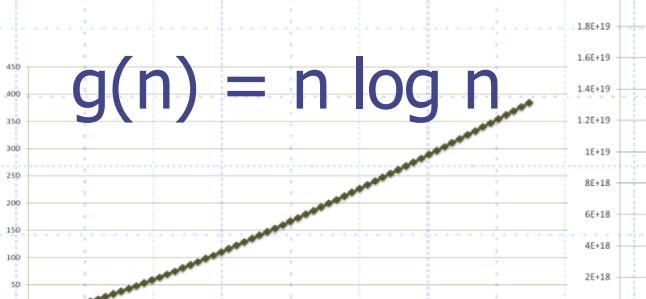
- En un gráfico logarítmico podemos ver el cambio en la tasa de crecimiento



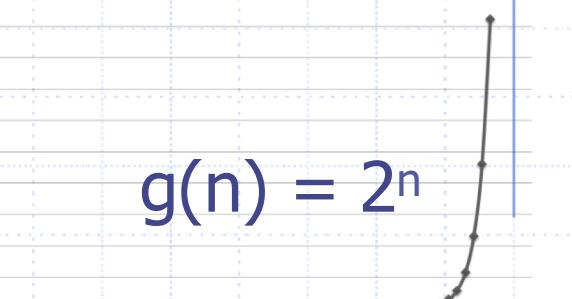
Funciones Graficadas utilizando una escala “Normal”



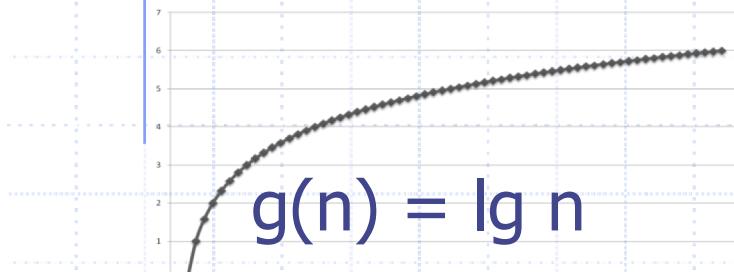
$$g(n) = 1$$



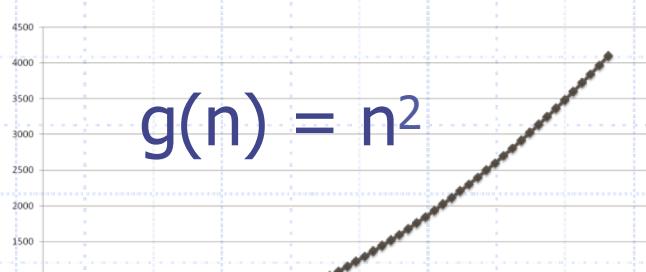
$$g(n) = n \log n$$



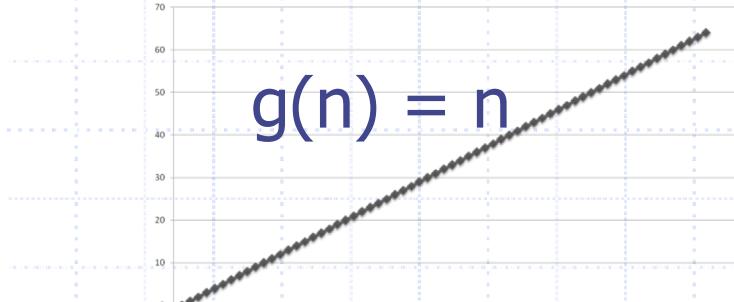
$$g(n) = 2^n$$



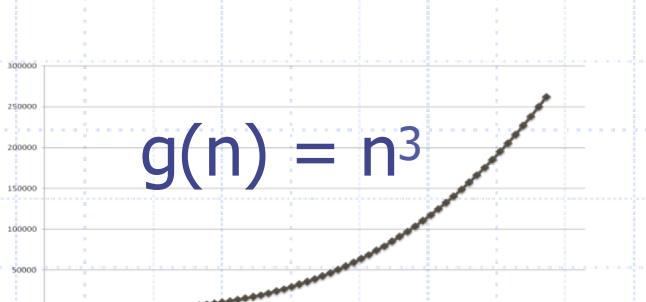
$$g(n) = \lg n$$



$$g(n) = n^2$$



$$g(n) = n$$



$$g(n) = n^3$$

Operaciones Primitivas



- Computaciones básicas realizados por un algoritmo.
- Identifiable en pseudocódigo
- En gran medida independiente del lenguaje de programación.
- La definición exacta no es importante
- Se supone que tarda un constante en un modelo de computador teórico.
- Ejemplos:
 - Evaluar una expresión
 - Asignar un valor a una variable
 - Indexación en una matriz
 - Llamando a un método
 - Regresar de un método

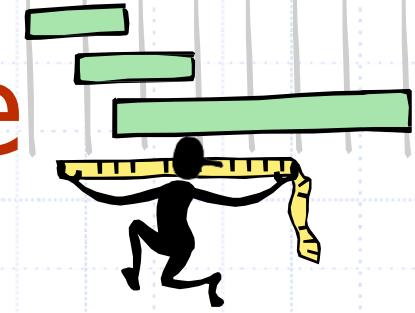
Cant. de Operaciones Primitivas

- Ejemplo: Inspeccionando el pseudocódigo, podemos determinar el número máximo de operaciones primitivas ejecutadas por un algoritmo, en función del tamaño de la entrada

Algorithm arrayMax(A, n):
Input: An array A storing $n \geq 1$ integers.
Output: The maximum element in A .

```
currentMax ←  $A[0]$ 
for  $i \leftarrow 1$  to  $n - 1$  do
    if  $currentMax < A[i]$  then
         $currentMax \leftarrow A[i]$ 
return  $currentMax$ 
```

Estimación del Tiempo de Ejecución



- El algoritmo **arrayMax** ejecuta $7n - 2$ operaciones primitivas en el peor de los casos, $5n$ en el mejor de los casos. Podemos definir:

a = Tiempo que tarda la operación primitiva más rápida

b = Tiempo que tarda la operación primitiva más lenta

- Sea $T(n)$ el tiempo en el peor de los casos de **arrayMax**. Entonces,

$$a(5n) \leq T(n) \leq b(7n - 2)$$

- Por lo tanto, el tiempo de ejecución $T(n)$ está acotado por dos funciones lineales

Por qué la es Importante?

tiempo de ejecución	time for $n + 1$	time for $2n$	time for $4n$
$c \lg n$	$c \lg (n + 1)$	$c (\lg n + 1)$	$c(\lg n + 2)$
cn	$c(n + 1)$	$2cn$	$4cn$
$c n \lg n$	$\sim c n \lg n + cn$	$2c n \lg n + 2cn$	$4c n \lg n + 4cn$
$c n^2$	$\sim c n^2 + 2cn$	$4c n^2$	$16c n^2$
$c n^3$	$\sim c n^3 + 3cn^2$	$8c n^3$	$64c n^3$
$c 2^n$	$c 2^{n+1}$	$c 2^{2n}$	$c 2^{4n}$

tiempo de cuadriplica cuando el tamaño del problema solo es el doble



Análisis de Algoritmos Recursivos

- Usamos una función, $T(n)$, para derivar una **relación de recurrencia** que caracterice el tiempo de ejecución del algoritmo en términos de valores más pequeños de n

Algorithm recursiveMax(A, n):

Input: An array A storing $n \geq 1$ integers.

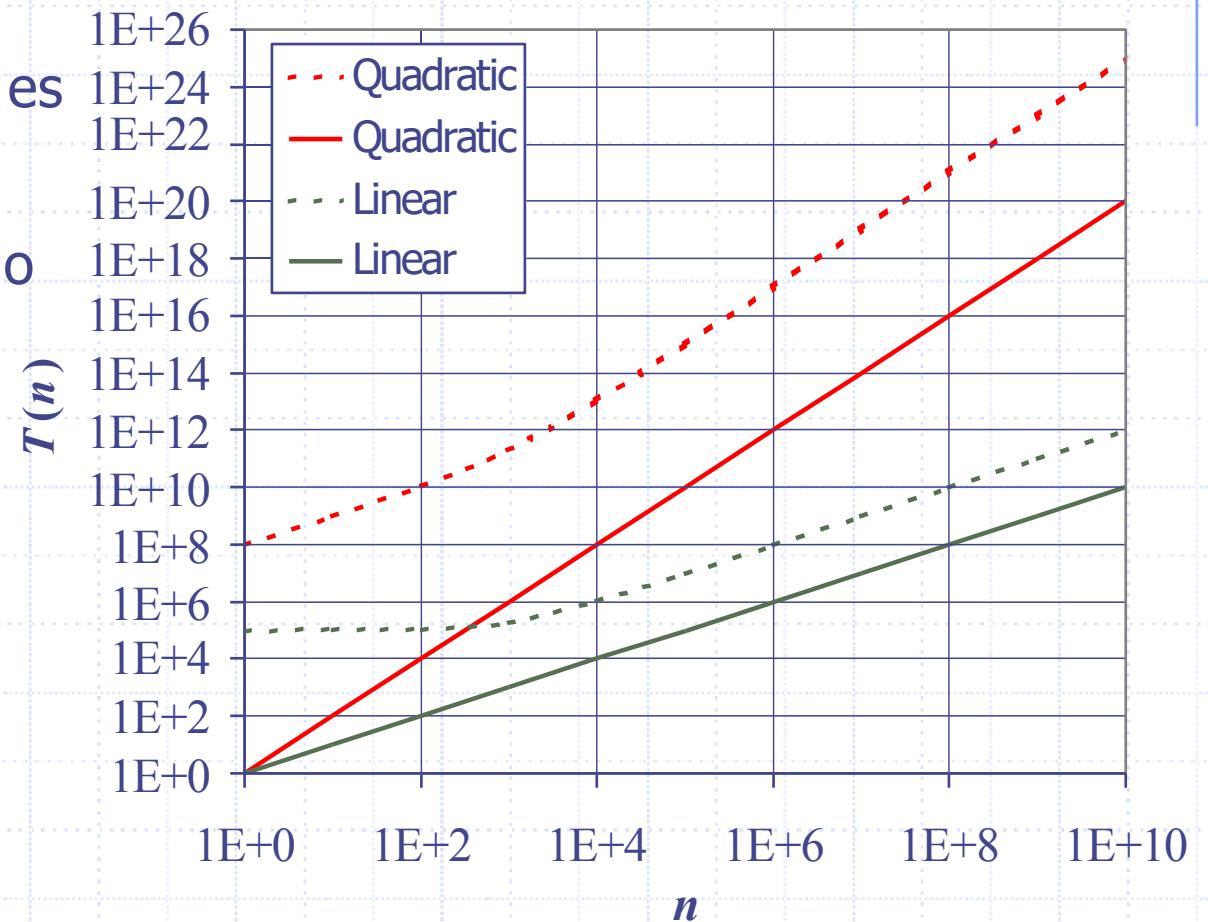
Output: The maximum element in A .

```
if  $n = 1$  then  
    return  $A[0]$   
return max{recursiveMax( $A, n - 1$ ),  $A[n - 1]$ }
```

$$T(n) = \begin{cases} 3 & \text{if } n = 1 \\ T(n - 1) + 7 & \text{otherwise,} \end{cases}$$

Factores Constantes

- La tasa de crecimiento es poco afectado por:
 - factores constantes, o
 - términos de orden inferior
- Ejemplos
 - $10^2 n + 10^5$ es una función lineal
 - $10^5 n^2 + 10^8 n$ es un función cuadrática



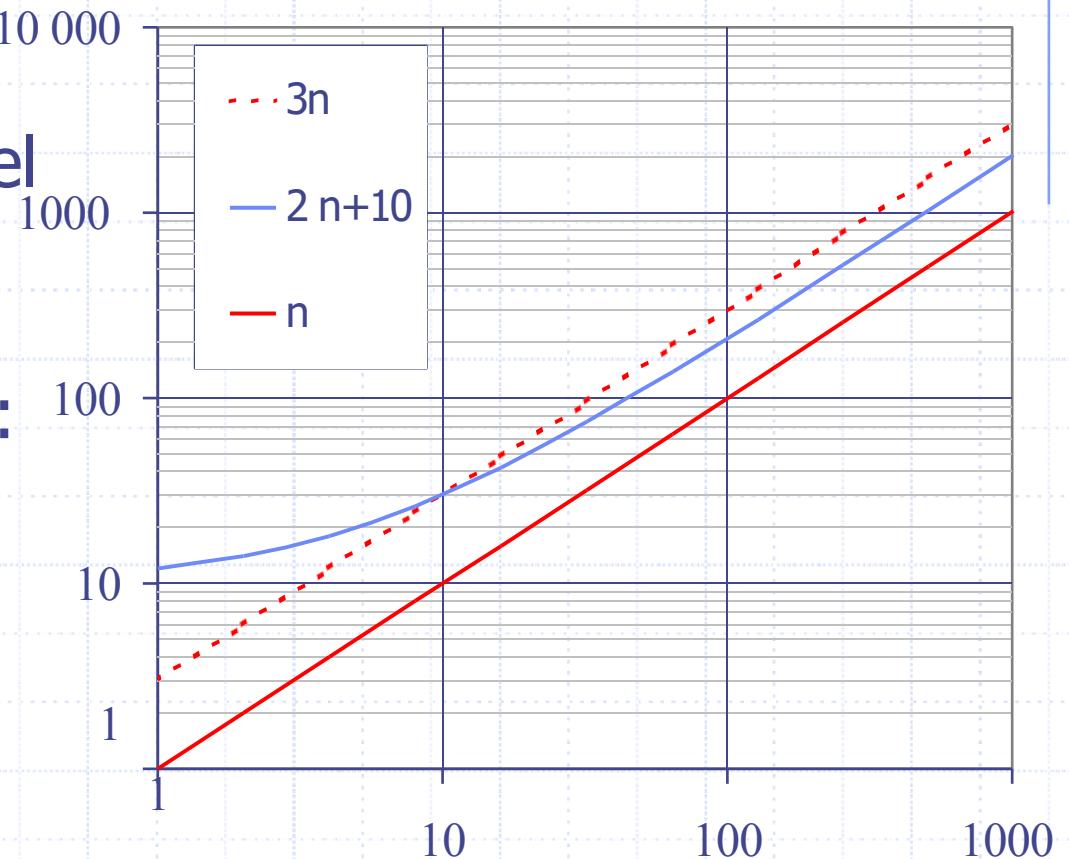
Notación Big-Oh

Dadas dos funciones $f(n)$ y $g(n)$, decimos que $f(n)$ es del orden de $g(n)$, escribimos, $O(g(n))$ si hay positivos constantes c y n_0 tales que:

$$f(n) \leq c g(n) \text{ para } n \geq n_0$$

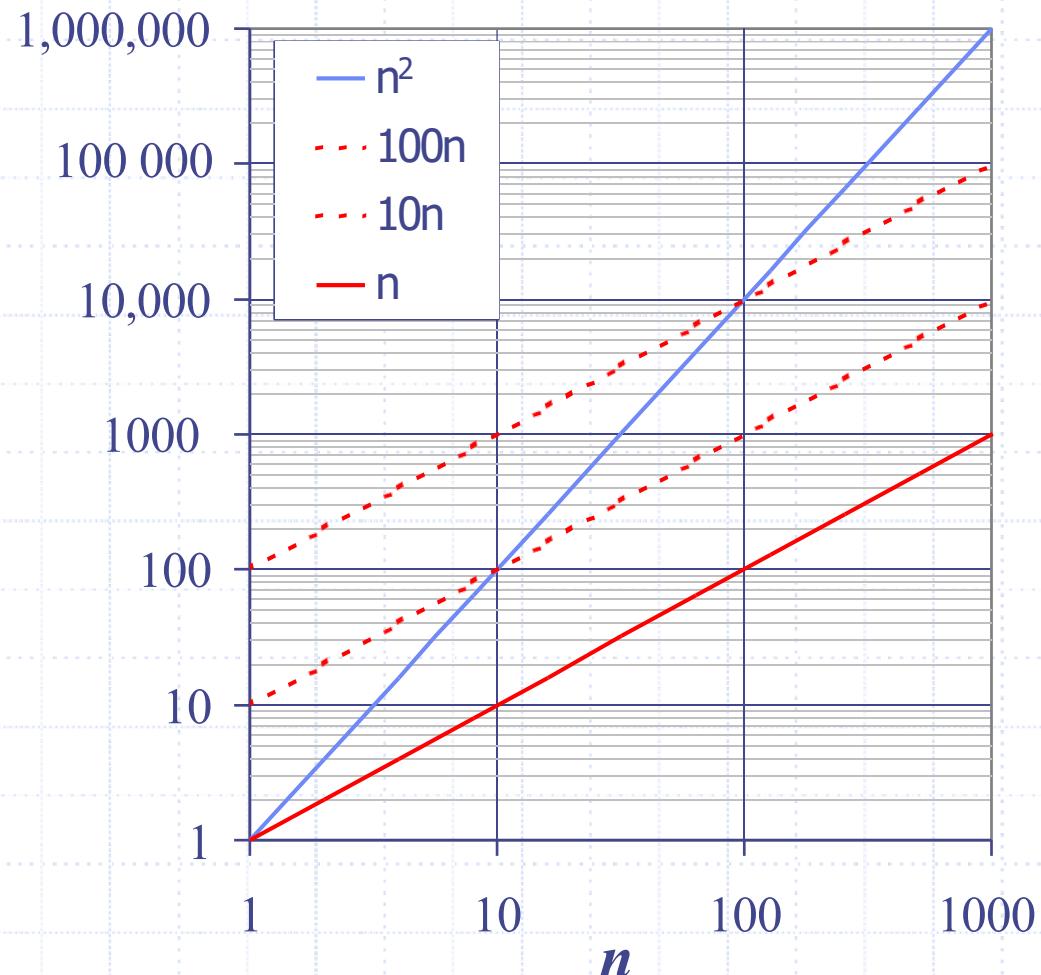
□ Ejemplo: $2n+10$ es $O(n)$

- $2n + 10 \leq cn$
- $(c - 2)n \geq 10$
- $n \geq 10/(c - 2)$
- Elijimos $c = 3$ y $n_0 = 10$



Ejemplo Big-Oh

- Ejemplo: la función n^2 no es $O(n)$
 - $n^2 \leq cn$
 - $n \leq c$
 - Esta desigualdad no puede ser cumplida, entonces c debe ser una constante



Más Ejemplos Big-Oh



$7n - 2$

$7n - 2$ es $O(n)$

necesita $c > 0$ y $n_0 \geq 1$ tal que $7n - 2 \leq c n$, para $n \geq n_0$

esto es cierto para $c = 7$ y $n_0 = 1$

$3n^3 + 20n^2 + 5$

$3n^3 + 20n^2 + 5$ es $O(n^3)$

necesita $c > 0$ y $n_0 \geq 1$ tal que $3n^3 + 20n^2 + 5 \leq c n^3$ para $n \geq n_0$

esto es cierto para $c = 4$ y $n_0 = 21$

$3 \log n + 5$

$3 \log n + 5$ es $O(\log n)$

necesita $c > 0$ y $n_0 \geq 1$ tal que $3 \log n + 5 \leq c \log n$ para $n \geq n_0$

esto es cierto para $c = 8$ y $n_0 = 2$

Big-Oh y la Tasa de Crecimiento

- La notación de Big-Oh nos da un límite superior para la tasa de crecimiento de una función
- Decir “ $f(n)$ es $O(g(n))$ ”, significa que el la tasa de crecimiento de $f(n)$ es menor que la tasa de crecimiento de $g(n)$
- Podemos usar la notación Big-Oh para clasificar funciones según su tasa de crecimiento

	$f(n)$ es $O(g(n))$	$g(n)$ es $O(f(n))$
$g(n)$ crece más	SI	No
$f(n)$ crece más	No	SI
Mismo crecimiento	SI	SI

Reglas de la Notación Big-Oh



- Si es $f(n)$ un polinomio de grado d , entonces $f(n)$ es $O(n^d)$, es decir,
 1. Eliminamos los términos con menor grado
 2. Eliminamos los factores constantes
- Usar la clase más pequeña posible de funciones
 - "2n es $O(n)$ " en lugar de " 2n es $O(n^2)$ "
- Usar la expresión más simple de la clase
 - " $3n+ 5$ es $O(n)$ " en lugar de " $3n+ 5$ es $O(3n)$ "

Análisis Asintótico

- El análisis asintótico de un algoritmo determina el tiempo de ejecución en big-Oh notación
- Para realizar el análisis asintótico
 - Encontramos el peor número de operaciones primitivas ejecutadas como una función sobre el tamaño de la entrada
 - Expresamos esta función con big-Oh notación
- Ejemplo:
 - Decimos que el algoritmo `arrayMax` “se ejecuta en tiempo $O(n)$ ”
- Ya que los factores constantes y los términos de orden inferior eventualmente se eliminan de todos modos, podemos ignorarlos al contar operaciones primitivas

Conceptos Matemáticos a Revisar



- Sumatorias
- Potenciación
- Logaritmos
- Pruebas Técnicas

- Propiedades de potencias:

$$a^{(b+c)} = a^b a^c$$

$$a^{bc} = (a^b)^c$$

$$a^b / a^c = a^{(b-c)}$$

$$b = a^{\log_a b}$$

$$b^c = a^{c * \log_a b}$$

- Propiedades de los logaritmos:

$$\log_b(xy) = \log_b x + \log_b y$$

$$\log_b(x/y) = \log_b x - \log_b y$$

$$\log_b x a = a \log_b x$$

$$\log_b a = \log_x a / \log_x b$$